Claims

1. (Previously Presented) A method of type-checking a code segment written in a programming language comprising:

translating the code segment from the programming language to one or more representations of an intermediate language, wherein the one or more representations of the intermediate language are capable of representing programs written in a plurality of different source languages, wherein the plurality of different source languages comprise at least one typed source language and at least one untyped source language; and

type-checking the one or more representations based on a rule set, wherein the rule set comprises rules for type-checking a type designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known.

2. (Canceled)

3. (Original) The method of claim 1 wherein the rule set is selected from a plurality of rule sets.

4. (Previously Presented) The method of claim 3 wherein only a fraction of the plurality of rule sets contain rules for type-checking a type designated as an unknown type, wherein the unknown type indicates that an element of the representation is of a type that is not known.

5. (Original) The method of claim 1 wherein the rule set further comprises rules for type-checking types representing categories of types found in a plurality of programming languages.

6. (Previously Presented) A method of selectively retaining type information during compilation in a code segment written in a programming language, the method comprising:

translating the code segment from the programming language to one or more representations of an intermediate language;

for each representation, determining whether to retain type information for one or more elements of the representation;

based on the determination, associating one or more elements of the representation with a type, designated as an unknown type, indicating the element can be of any type; and

type-checking the one or more representations based on a rule set, wherein the rule set comprises rules for type-checking the type designated as the unknown type.

7. (Original) The method of claim 6 wherein the determination is based on a current stage of compilation, a characteristic of each representation, or the programming language.

8. (Canceled)

9. (Previously Presented) The method of claim 6 wherein the type, designated as the unknown type, indicating the element can be of any type has size information associated with it.

10. (Previously Presented) The method of claim 9 further comprising generating code from at least elements associated with the type, designated as the unknown type, indicating the element can be of any type based on the size information.

11. (Canceled)

12. (Previously Presented) A method of translating types associated with a plurality of programming languages to types of an intermediate language, the method comprising:

replacing the types associated with the plurality of programming languages with the types of the intermediate language, wherein the types of the intermediate language comprise general categories of the types associated with the plurality of programming languages and a type designated as an unknown type, wherein the type designated as the unknown type has size information associated with it, wherein the size information comprises size information of a machine representation of the type designated as the unknown type.

13. (Original) The method of claim 12 wherein the types of the intermediate language further comprise types related to programming language specific primitive types.

14. **(Currently Amended)** A computer system for type-checking an intermediate representation of source code in a compiler comprising:

a computer-readable storage medium containing one or more types associated with elements of the intermediate representation, wherein at least one of the types, designated as an unknown type, indicates an element can be of any type;

a computer-readable storage medium containing one or more rule sets comprising rules associated with the type, designated as the unknown type, indicating an element can be of any type; and

a type-checker module, wherein the type-checker is configured for applying the one or more rule sets to the elements of the intermediate representation.

15. (Previously Presented) The system of claim 14 wherein the type, designated as the unknown type, indicating the element can be of any type has size information associated with it.

16. (Canceled)

17. (Original) The system of claim 14 wherein the one or more rule sets applied to the elements of the intermediate representation are selected based on the stage of compilation.

18. (Original) The system of claim 14 wherein the one or more rule sets applied to the elements of the intermediate representation are selected based on a characteristic of the source code.

19. (Original) The system of claim 14 wherein the one or more rule sets applied to the elements of the intermediate representation are selected based on the intermediate representation.

20. (Previously Presented) The system of claim 14 wherein only a fraction of the one or more rule sets contain rules for type-checking a type, designated as an unknown type, that indicates an element can be of any type.

21. (Original) The system of claim 14 wherein the one or more rule sets further comprise rules for type-checking types representing categories of types found in a plurality of programming languages.

22. (Original) The system of claim 14 wherein the system selectively retains type information for some elements of the intermediate representation and selectively does not retain type information for other elements of the intermediate representation.

23. (Previously Presented) The system of claim 22 wherein the system selectively does not retain type information for an element of the intermediate representation by replacing a type associated with the element with the type, designated as the unknown type, indicating an element can be of any type.

24. (Previously Presented) A method of representing types in an intermediate language comprising:

defining a plurality of types to be associated with elements of the intermediate language, wherein one of the plurality of types indicates that an element of the intermediate language is associated with a type designated as an unknown type;

wherein the type indicating that an element of the intermediate language is associated with the type designated as the unknown type has a size associated with it, wherein the size represents size of a machine representation of the type designated as the unknown type.

25. (Canceled)

26. (Previously Presented) The method of claim 24 wherein an element of the intermediate language that was previously associated with another type is associated with the type indicating that the element is associated with the type designated as the unknown type.

27. (Original) The method of claim 24 wherein the plurality of types further comprises types representing categories of types found in a plurality of programming languages.

28. (Original) A computer-readable storage medium containing computer-executable instructions for implementing the method of claim 24.

29. (Original) A computer-readable storage medium containing computer-executable instructions for implementing the method of claim 1.

30. (Previously Presented) The method of claim 1 wherein the rule set further comprises rules for dropping type information for one or more elements of the representation by changing a known type of the one or more elements to the type designated as the unknown type.

31. (Previously Presented) The method of claim 6 wherein at least one of the one or more representations of the intermediate language supports dropping type information by designating a type as an unknown type.

32. (Previously Presented) The system of claim 15 wherein the size information comprises size information of a machine representation of the type designated as the unknown type.